Endpoint Detection and Response User Manual

Course:

Cyber Security, May, 2020

Project Name:

#3 - EDR.

Objective:

Create an Endpoint Detection and Response system.

Student Name:

Robert Yonatan Tiger.

Endpoint Detection and Response User Manual

Table of Contents:

Endpoint Detection and Response User Manual

Introduction

What is EDR (Endpoint Detection and Response) system

Endpoint Detection and Response (EDR), also known as **Endpoint Threat Detection and Response** (ETDR) is a cyber technology that continually monitors and responds to mitigate cyber threats.

Endpoint Detection and Response (EDR) is a technology used to protect endpoints, which are computer hardware devices, from threat. Creators of the EDR technology-based platforms deploy tools to gather data from endpoint devices, and then analyse the data to reveal potential cyber threats and issues. It is a protection against hacking attempts and stealing of user data. The software is installed on the end-user device and it is continually monitored. The data is stored in a centralized database. In an incident when a threat is found, the end-user is immediately prompted with preventive list of actions

Code Workflow

1. Runs a listening server on a Linux machine.
2. Clients connecting to the server and sends data to the server.
3. Server logs relevant data to log files.
4. Meanwhile, the server monitors if a client has been disconnected and alerts to the screen.

Requirements

Server Side:

- Server running Linux.
- Root privileges user.
- Python 3+.

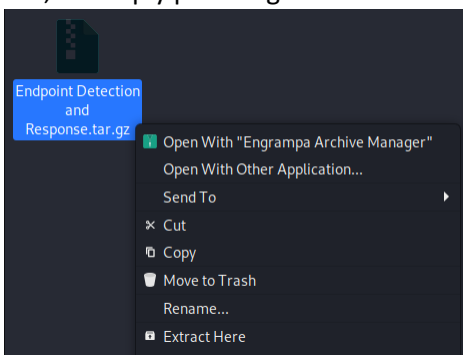
Client Side:

- Any client machine running Windows or Linux.
- Connect to the same local network of the server.
- Internet Connection.
- Python 3+.
- Scapy module
- Bs4 module.
- Lxml.

Endpoint Detection and Response User Manual

Starting the EDR from 0

1. Extract the files to a folder, or simply press Right Click mouse button → Extract Here.



2. Choose a port in the Server.py file to run host the server on:

```
PORT = 1111  # Modify to your liking (preffered 1000+)
```

Figure I: Server.py

3. Set the server's IP address as HOST and the port you chose in Server.py:

```
HOST = '10.100.102.65'  # Server IP.
PORT = 1111  # Server's listening port.
```

Figure II: Client.py

4. Open a terminal in the EDR path, run the Server.py and wait for connections from clients:

```
n and Response# python3 Server.py
[INFO] Apache2 Server Started (http://localhost:80)
[INFO] restricted_sites.html copied to /var/www/html
Edit the file inside /var/www/html to add or remove restricted sites for clients.
[INFO]Server address binded to self (0.0.0.0)
[INFO] Listening on port 1111... (Waiting for connections)
```

Figure III: Server.py: First run.

5. Copy the Client.py to all clients you want to monitor. (Make sure they are on the same network) and run Client.py to connect and start the monitoring process:

```
and Response# python3 Client.py
Trying to connect to the server...
[INFO] You are connected to: 10.100.102.66 in port: 1111.
Successfully connected to EDR Server at 0.0.0.0:1111
```

Figure IV: Client.py: First run.

6. You have successfully connected a client. Now do it for all other clients in your network.

Endpoint Detection and Response User Manual

Breaking Down the Program

Code Intro and Objective Setting

```
3   ================================================================
4   Course:
5       Cyber Security, May, 2020
6   Project Name:
7       #3 - Python - Endpoint Detection and Response.
8   Objective:
9       Create an Endpoint Detection and Response System (EDR)
10  Student Name:
11      Robert Jonny Tiger.
12  ================================================================
```

Figure V: Lines 3-12: Basic information, The objective of the program.

Imports

Server Side:

```
import socket
import urllib.request
from pathlib import Path
from subprocess import check_output, run
from threading import Thread
from time import sleep
```

Client Side:

```
import socket
import urllib.request
from os import path, remove
from platform import system
from subprocess import check_output, run
from threading import Thread
from time import sleep
from bs4 import BeautifulSoup
from scapy.all import *
```

Endpoint Detection and Response User Manual

Start of the Program

Server Side:

```
174    # Start of the Script:
175    if __name__ == '__main__':
176        apache2Start()
177        main()
```

Figure VI: Line 175: Start of the program.

Workflow described is: First start an apache2 server, then run the rest of the function. Without apache2 half of the program is not useable. (More info in Functions – Server Side:)

Client Side:

```
161    if __name__ == '__main__':
162        main()
163        Thread(target=restricted_Sites_List_Maker).start()
164        Thread(target=MITM).start()
165        Thread(target=sniff(prn=findDNS)).start()
```

Figure VII: Line 161: Start of the program.

Workflow described is: start the main function, then start 3 threads – each responsible of completing certain objects. (More info in Functions – Client Side:)

Functions

Server Side:

main

- Binds socket to `((HOST, PORT))`, listening to connections, accepting new connections, sets a format for `connName`.
- Closes all previous connections if Server.py restarted
- Accepts new connections in `while True`.
- Sends welcome message to new clients, appends new client's socket objects and `connName` to the lists.
- Starts 2 threads: One for handling clients and the other for checking connections with clients.
- Lists active connections count after a new client has connected.

Endpoint Detection and Response User Manual

```
64  def main():
65      try:
66          serverSocket.bind((HOST, PORT))  # Bind the socket.
67          print(f'[INFO]Server address binded to self ({HOST})')
68      except socket.error as error:
69          exit(
70              f'[ERROR] Error in Binding the Server:\n\033[31m{error}\033[0m')
71      print(
72          f'[INFO] Listening on port {PORT}... (Waiting for connections)')
73      serverSocket.listen(50)
74      for clientSocket in openClientSocketsList:
75          # Closes all preavious connections if Server.py restarted:
76          clientSocket.close()
77          # Deletes all previous open client sockets and active addresses from the lists:
78          del openClientSocketsList[:], activeAddressesList[:]
```

```
80      while True:
81          try:
82              # Accepts connections:
83              conn, (address, port) = serverSocket.accept()
84              # Appends the client's socket to the list:
85              openClientSocketsList.append(conn)
86              # Set a format for the connName using client's address and port:
87              connName = '{}:{}'.format(address, port)
88              print(f'[INFO] {connName} Connected!')
89              welcomeMessage = f'Successfully connected to EDR Server at {HOST}:{PORT}'
90              # Sends welcome message to the client:
91              conn.send(welcomeMessage.encode())
92              global connectionsCount
93              connectionsCount += 1  # Adding +1 to the connections count.
94              # Appends the new address to the activeAddressesList:
95              activeAddressesList.append(connName)
96              # Prints current connections count:
97              print(
98                  f'[INFO] Number of Active Connections: {connectionsCount}')
99              # Starts a new thread to handle each client (args are the connection and formatted connection name):
100             Thread(target=handleClient, args=(conn, connName)).start()
101             # Starts a checkConnections thread:
102             Thread(target=checkConnections).start()
103         except socket.error as acceptError:
104             print(
105                 f'[ERROR] Accepting Connection from: {conn.getpeername()}:\n\033[31m{acceptError}\033[0m')
106             continue
```

Figure VIII: Lines 64-106: main function.

➔ **Output:**

```
and Response# python3 Server.py
[INFO] Apache2 Server Started (http://localhost:80)
[INFO] restricted_sites.html copied to /var/www/html
Edit the file inside /var/www/html to add or remove restricted sites for clients.
[INFO]Server address binded to self (0.0.0.0)
[INFO] Listening on port 1111... (Waiting for connections)
[INFO] 10.100.102.66:36730 Connected!
[INFO] Number of Active Connections: 1
```

Apche2start

- Checks if apache2 is installed. if not - exits the code with a message.
- Starts apache2 server.

Endpoint Detection and Response User Manual

- Copies the local restriced_sites.html to the actual html folder in /var/www/html where apache2 is running from.
- Server admin edits the file inside the /var/www/html to update the restricted sites list.

```
42  def apache2Start():
43      apache2InstallStatus = check_output(
44          "dpkg --get-selections | grep apache2-bin | awk '{print $2}'", shell=True)
45      if apache2InstallStatus:
46          run("service apache2 start", shell=True)
47          try:
48              response = urllib.request.urlopen(
49                  'http://0.0.0.0/restricted_sites.html')
50          except:
51              while response.status != 200:
52                  run('service apache2 restart', shell=True)
53                  sleep(5)
54      else:
55          exit('[ERROR] \033[31mApache2 service is not installed, make sure to install Apache2 and run the server again\033[0m')
56      print('[INFO] Apache2 Server Started (http://localhost:80)')
57      print('[INFO] restricted_sites.html copied to /var/www/html\nEdit the file inside /var/www/html to add or remove restricted
        sites for clients.')
```

Figure IX: Lines 42-57: apache2Start function

➔ **Output:**

```
n and Response# python3 Server.py
[INFO] Apache2 Server Started (http://localhost:80)
[INFO] restricted_sites.html copied to /var/www/html
Edit the file inside /var/www/html to add or remove restricted sites for clients.
```

handleClient(conn, connName)

- Main function to receive data from all clients.
- Handles client connections using args from main.
- If data has "MAC" in it, logs the data to `'MitM Logger.log'`
- If data has "`restricted`" in it, logs the data to '`Restricted Sites Logger.log`'

Endpoint Detection and Response User Manual

```
114  def handleClient(conn, connName):
115      while True:
116          try:
117              data = conn.recv(4096).decode()
118              if "MAC" in data:
119                  # Timestamp for the log file:
120                  timestamp = check_output(
121                      "date -u +'%d/%m/%Y %H:%M'", shell=True).decode().rstrip()
122                  print(
123                      '[WARNING] Possible Man in the Middle attack. Check MitM Logger.log')
124                  with open(f"{PROJECTPATH}/MitM Logger.log", "a+") as MitMLog:
125                      MitMLog.write(
126                          f"[{timestamp}]{TAB_1}[{connName}]:\n{data}")  # Logs the MitM attack from the c
127
128              if "restricted" in data:
129                  # Timestamp for the log file:
130                  timestamp = check_output(
131                      "date -u +'%d/%m/%Y %H:%M'", shell=True).decode().rstrip()
132                  print(
133                      f'[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log')
134                  with open(f'{PROJECTPATH}/Restricted Sites Logger.log', 'a+') as restrictedLog:
135                      restrictedLog.write(
136                          f"[{timestamp}]{TAB_1}[{connName}]:\n{data}")  # Logs the restricted site from t
                         Sites Logger.log'
137          except:
138              pass
```

Figure X: Lines 114-138: handleClient function.

➔ **Output:**

```
[WARNING] Possible Man in the Middle attack. Check MitM Logger.log
[WARNING] Possible Man in the Middle attack. Check MitM Logger.log
[WARNING] Possible Man in the Middle attack. Check MitM Logger.log
[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log
[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log
[ALERT] Someone entered to a restricted site. Check Restricted Sites Logger.log
```

The server knows where the data came from and logs its details in the log with the current timestamp.

Appends data to 'MitM Logger.log':

MitM Logger.log Client.py

```
[31/08/2020 09:08]  [10.100.102.65:40428]:
[WARNING]Found MAC address duplication. Possible Man in the Middle Attack!
Check this MAC: 8c:59:c3:ee:da:f5

[WARNING]Found MAC address duplication. Possible Man in the Middle Attack!
Check this MAC: 00:0c:29:7a:f1:37
```

Figure XI: MitM Logger.log

Appends data to 'Restricted Sites Logger.log':

Endpoint Detection and Response User Manual

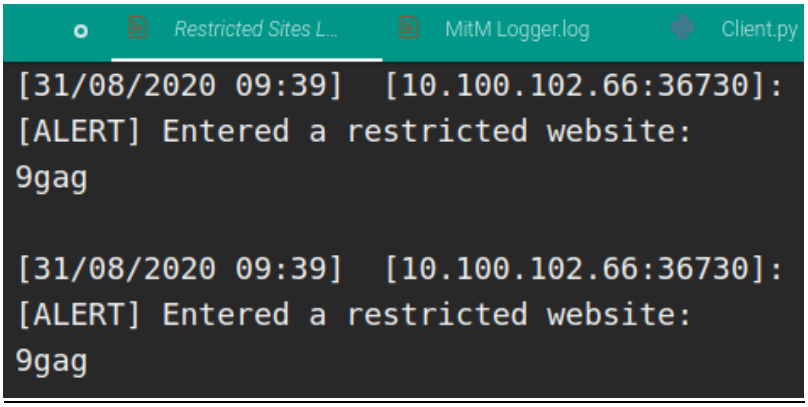


Figure XII: Restricted Sites Logger.log

checkConnections

- Checks which clients are alive by iterating through every client socket object and trying to send a whitespace string.
- If an exception occurs, it means that the client is dead.
- Deletes the client socket object and address from the lists and decreasing 1 from connections count.
- Prints number of current connections.
- Prints a list of the connections if there are any left.
- This check happens every 30 seconds.

```
147  def checkConnections():
148      while True:
149          global connectionsCount
150          if len(openClientSocketsList) != 0:
151              for x, currentSocket in enumerate(openClientSocketsList):
152                  try:
153                      # Send a whitespace to every socket in the list:
154                      pingToClientMessage = ' '
155                      currentSocket.send(pingToClientMessage.encode())
156                  except:
157                      print(f'[INFO] Client {x} Disconnected!')
158                      # Deletes the client socket and address from the lists:
159                      del openClientSocketsList[x], activeAddressesList[x]
160                      connectionsCount -= 1
161                      if connectionsCount == 0:  # If no connections left:
162                          print(f'[INFO] No active connections left.')
163                      else:  # If there are still connections left:
164                          print(
165                              f'[INFO] Number of Active Connections: {connectionsCount}')
166                          print('[INFO] Active addresses connected:')
167                          # Prints a list of the current open connections:
168                          for index, value in enumerate(activeAddressesList):
169                              print(f'{TAB_1}{index}.{TAB_1}{value}')
170                      continue
171          sleep(30)
```

Figure XIII: Lines 147-171: checkConnections function

➔ **Output:**

```
[INFO] Client 0 Disconnected!
[INFO] Number of Active Connections: 1
[INFO] Active addresses connected:
        0.      10.100.102.66:37836
[INFO] Client 0 Disconnected!
[INFO] No active connections left.
```

Client Side:

main:

- Creates a socket object.
- Connects to server and prints the welcome message.

```
30  def main():
31      global clientSocket
32      # Client's Socket Object:
33      clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
34
35      print('Trying to connect to the server...')
36      try:
37          clientSocket.connect((HOST, PORT))  # Connects to the server's socket.
38          print(f'[INFO] You are connected to: {HOST} in port: {PORT}.')
39          welcomeMessage = clientSocket.recv(1024)  # Receives welcome message.
40          print(welcomeMessage.decode())
41      except socket.error as error:
42          exit(
43              f'[ERROR] Connecting to the server failed:\n\033[31m{error}\033[0m')
```

Figure XIV: Lines 30-43: main function

MitM:

- Checks for duplications in ARP table in both Linux and Windows.
- Iterates through the MAC addresses in the ARP table, adding them to a list.
- If a duplication occurs - the value of the MAC in the dictionary will rise by 1.
- For every MAC key that has a value of more than 1, it will send a warning message to the server.
- The scan happens every `sleep(x seconds)` - modify to your liking.

Endpoint Detection and Response User Manual

For Windows OS:

```
52  def MITM():
53      while True:
54          macList = []
55          macDict = {}
56          if runningOS == "Windows":
57              ARPmacs = check_output("arp -a", shell=True).decode()
58
59              for line in ARPmacs.splitlines():
60                  if "dynamic" in line:
61                      macList.append(line[24:41])
62
63              for MAC in macList:
64                  if MAC in macDict:
65                      macDict[MAC] = macDict[MAC] + 1
66                  else:
67                      macDict[MAC] = 1
68
69              for MAC, value in macDict.items():
70                  if value >= 2:
71                      clientSocket.send(
72                          f'[WARNING] Found MAC address duplication. Possible Man in the Middle Attack!\nCheck this MAC:
 •                        {MAC}\n\n'.encode())
```

Figure XV: Lines 52-72: MitM function for Windows

For Linux OS:

```
74          elif runningOS == "Linux":
75              ARPmacs = check_output(
76                  "arp | awk '{print $3}' | grep -v HW | grep -v eth0", shell=True).decode()
77              for line in ARPmacs.splitlines():
78                  macList.append(line)
79
80              for MAC in macList:
81                  if MAC in macDict:
82                      macDict[MAC] = macDict[MAC] + 1
83                  else:
84                      macDict[MAC] = 1
85              for MAC, value in macDict.items():
86                  if value >= 2:
87                      clientSocket.send(
88                          f'[WARNING]Found MAC address duplication. Possible Man in the Middle Attack!\nCheck this MAC:
 •                        {MAC}\n\n'.encode())
89          sleep(15)
```

Figure XVI: Lines 74-89: MitM function for Linux

➔ **Output:**

```
188    [31/08/2020 20:26]  [10.100.102.67:37300]:
189    [WARNING]Found MAC address duplication. Possible Man in the Middle Attack!
190    Check this MAC: 8c:59:c3:ee:da:f5
```

The client-side code is responsible on sending the warning message with the duplicated MAC that will be logged to the log file.

Endpoint Detection and Response User Manual

restricted_Sites_List_Maker:

- Creates a list of website names that will be used as arguments for the DNS sniffer.
- The function gets the websites from the restricted_sites.html webpage running on the apache2 server.
- Only the server admin will have access to the html where the blacklist is stored.
- The update happens every `sleep(x seconds)` - modify to your liking.

```
97  def restricted_Sites_List_Maker():
98      while True:
99          # Restricted Websites webpage:
100         restrictedWebsites = f"http://{HOST}/restricted_sites.html"
101
102         HTMLrestrictedWebsites = urllib.request.urlopen(
103             restrictedWebsites).read()
104         soup = BeautifulSoup(HTMLrestrictedWebsites, features="lxml")
105
106         textRestictedWebsites = soup.body.get_text()  # Gets text.
107
108         # Breaks into lines and remove leading and trailing space on each:
109         lines = (line.strip() for line in textRestictedWebsites.splitlines())
110
111         # Breaks multi-headlines into a line each:
112         chunks = (phrase.strip()
113                   for line in lines for phrase in line.split("  "))
114
115         # Drops blank lines. Final result:
116         textRestictedWebsites = '\n'.join(chunk for chunk in chunks if chunk)
```

Figure XVII: Lines 97-116: Gets clear website name to add to the list

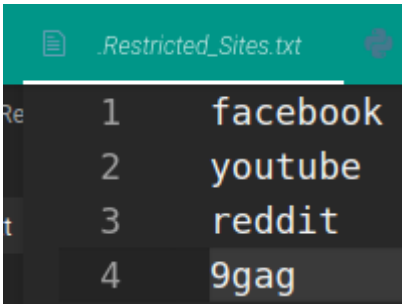
Endpoint Detection and Response User Manual

```
118         # Creates \ Overwrites \ the list of sites to a txt file from the html page.
119         if runningOS == "Windows":
120             if path.exists("Restricted_Sites.txt"):
121                 remove("Restricted_Sites.txt")
122 
123             with open("Restricted_Sites.txt", "w") as restrictedSitesFile:
124                 restrictedSitesFile.write(textRestictedWebsites)
125                 # Makes the file hidden.
126                 run("attrib +h Restricted_Sites.txt", shell=True)
127 
128             # Appends the site to the restrictedSitesList:
129             with open("Restricted_Sites.txt", "r") as f:
130                 for siteLine in f.readlines():
131                     restrictedSitesList.append(siteLine.strip())
132 
133         elif runningOS == "Linux":
134             with open(".Restricted_Sites.txt", "w") as restrictedSitesFile:
135                 restrictedSitesFile.write(textRestictedWebsites)
136 
137             # Appends the site to the restrictedSitesList
138             with open(".Restricted_Sites.txt", "r") as f:
139                 for siteLine in f.readlines():
140                     restrictedSitesList.append(siteLine.strip())
141         sleep(60)
```

Figure XVIII: Lines 118-141: restricted_Sites_List_Maker function

Creates a hidden file named 'Restricted_Sites.txt' and makes it hidden so that clients won't have access to modify it. Then, reads every line in the file and appends the line to the restricted sites list. Example: `[9gag, youtube, facebook]` → That will be the list of restricted sites that the sniffer will use as arguments.

➔ **Output:**



This is the file on the client side. The file is hidden and the list updates every x seconds.
The function iterates through the lines and adds every name of site to the `restrictedSitesList`.

Endpoint Detection and Response User Manual

findDNS:

- Sniffs DNS queries of the client.
- Gets only the name of the website from the query. Setting it to '`url`' variable.
- if the name of the site from the `restrictedSitesList` found in the current sniffed '`url`' variable - sends an alert to the server.

```
149  def findDNS(pkt):
150      if pkt.haslayer(DNS):
151          if "Qry" in pkt.summary():  # Only quearys.
152              # Gets only the name of the website from the queary:
153              url = pkt.summary().split('\"')[-2].replace("", "")[2:-2]
154              for site in restrictedSitesList:
155                  if site in url:
156                      clientSocket.send(
157                          f'[ALERT] Entered a restricted website:\n{site}\n\n'.encode())
```

Figure XIX: Lines 149-157: findDNS function

➔ **Output:**

```
1   [31/08/2020 20:48]  [10.100.102.67:37578]:
2   [ALERT] Entered a restricted website:
3   9gag
```

The client-side is responsible in sending an alert message with the restricted website to the server. The server logs the timestamps, address and port of the client and the message itself.